



rt
e
o
x
e

Développer des **services REST** en **Java**

Échanger des données au format **JSON**

Téléchargement
www.editions-eni.fr



Aurélie SOBRERO

Les éléments à télécharger sont disponibles à l'adresse suivante :

<http://www.editions-eni.fr>

Saisissez la référence ENI de l'ouvrage **EIREST** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Chapitre 1

Terminologie, rappels

1. SOA	11
2. ROA	12
2.1 L'interface uniforme	12
2.2 L'adressage	13
2.3 Le sans état	13
2.4 La connectivité	13
3. ROCA	13
3.1 Le côté serveur	14
3.2 Le côté client	15
4. JAXB	15
5. WADL	16

Chapitre 2

Introduction à JSON

1. Présentation de JSON	17
1.1 Règles d'écriture	17
1.1.1 Les clés	17
1.1.2 Les valeurs	17
1.1.3 Les couples clé/valeur	18
1.1.4 L'indentation	19
1.1.5 Les commentaires	19

2 ————— Développer des services REST

en Java - Échanger des données au format JSON

1.2	Comparaison avec XML.....	20
2.	Visualisation et validation d'un JSON	21
2.1	Le plugin JsonView.....	23
2.1.1	Coloration syntaxique	23
2.1.2	Détection d'erreurs de syntaxe	24
2.2	Le site JSONLint.....	26
3.	JSON et JavaScript	28
3.1	JavaScript et l'objet.....	28
3.2	La console JavaScript	31
3.2.1	Sous Mozilla Firefox.....	32
3.2.2	Sous Google Chrome	35
3.2.3	Sous Internet Explorer	36
3.3	Fonctions JavaScript utiles.....	38
3.3.1	JSON.parse	39
3.3.2	JSON.stringify	40
3.4	JavaScript : du DOM vers JSON	40
3.5	JavaScript et les erreurs de syntaxe	42
4.	JSON et Java, production et consommation	43
4.1	La bibliothèque Jsonic	45
4.1.1	JSON.encode et json.format	45
4.1.2	JSON.decode et json.parse.....	47
4.2	La bibliothèque Google Gson.....	50
4.2.1	gson.toJson	50
4.2.2	gson.fromJson.....	54

Chapitre 3

Introduction à REST

1.	Présentation de REST.....	57
2.	Une programmation stateless ou stateful ?	58
3.	REST vs RESTful	58

4. Méthodes HTTP	58
4.1 OPTIONS	60
4.2 HEAD et GET	61
4.2.1 Partial GET	61
4.2.2 Conditional GET	63
4.3 POST	64
4.3.1 Nouvelle ressource sans URL	64
4.3.2 Nouvelle ressource avec une URL	64
4.3.3 Pas de nouvelle ressource	64
4.4 PUT et PATCH	65
4.4.1 PUT	65
4.4.2 PATCH	66
4.5 LINK et UNLINK	66
4.6 DELETE	67
5. En-têtes HTTP	67
5.1 Accept-Charset, Accept-Encoding et Transfer-Encoding	67
5.1.1 Accept-Charset	68
5.1.2 Accept-Encoding	68
5.1.3 Transfer-Encoding	68
5.2 Accept-Language et Content-Language	69
5.3 Age	69
5.4 Allow	70
5.5 Authorization	70
5.6 Content-Length et Content-MD5	70
5.6.1 Content-Length	70
5.6.2 Content-MD5	71
5.7 Content-Location	71
5.8 Accept et Content-Type	72
5.8.1 Accept	72
5.8.2 Content-Type	72
5.9 Etag	73
5.10 If-Match et If-None-Match	73
5.11 If-Modified-Since et If-Unmodified-Since	73

4 ————— Développer des services REST

en Java - Échanger des données au format JSON

5.12 Content-Range, If-Range et Range	73
6. Formats de sortie et leurs types MIME	74
7. Format des URL, URL logiques et physiques	74
8. Liens vers les ressources	75
8.1 Action sur l'ensemble des ressources d'un type	75
8.2 Action sur une ressource donnée	76
8.3 Action sur une ressource liée	76
8.4 Format de retour	77
9. Gestion des exceptions	77
9.1 Codes HTTP d'erreurs	77
10. Test d'une requête REST	82
10.1 Sous Mozilla Firefox	82
10.2 Sous Google Chrome	83

Chapitre 4

Production de JSON avec Java et REST

1. Du POJO au JSON grâce aux annotations JAX-RS	85
1.1 Les bases	85
1.1.1 Choix de la méthode HTTP	85
1.1.2 Choix du chemin d'appel	86
1.1.3 Choix du type de retour	88
1.1.4 Choix du code HTTP de retour	92
1.2 @PathParam	94
1.3 @MatrixParam	96
1.4 @QueryParam	98
1.5 @FormParam	99
1.6 @HeaderParam	101
1.7 @CookieParam	102

2.	Implémentations de JAX-RS	104
2.1	Jersey	104
2.1.1	Création d'un projet fonctionnant avec Jersey	104
2.1.2	Transformation automatique d'un objet en JSON	105
2.1.3	Les annotations	107
2.1.4	Implémentation des services GET, POST, PUT et DELETE	119
2.1.5	Génération du WADL	130
2.2	Apache CXF	131
2.2.1	Création du projet	132
2.2.2	Création des services REST	134
2.2.3	Génération du WADL	135
2.2.4	Génération des JSON	137
2.2.5	Création d'une annotation pour gérer une nouvelle méthode HTTP	138
2.3	JBoss RESTEasy	142
2.3.1	Création du projet	142
2.3.2	Création du serveur	143
2.3.3	Exemples	144
2.3.4	Création des services REST	146
2.3.5	Génération du WADL	150
3.	Classes concrètes et interfaces	151
4.	Sécurité avec JAX-RS	153
4.1	Authentification	153
4.1.1	Apache CXF et l'authentification basique	153
4.1.2	Jersey et l'authentification basique	155
4.1.3	JBoss RESTEasy et l'authentification basique	158
4.2	Client Java	160
4.2.1	Apache CXF	160
4.2.2	Jersey	164
4.2.3	JBoss RESTEasy	166

6 ————— Développer des services REST

en Java - Échanger des données au format JSON

4.3	Autorisations	171
4.3.1	Apache CXF et Spring Security	171
4.3.2	Jersey	183
4.3.3	JBoss RESTEasy	187

Chapitre 5

Pour aller plus loin

1.	D'autres façons d'aborder REST	195
1.1	java.net.URL	195
1.2	Apache HttpComponents	198
1.3	Spring Data REST	200
1.3.1	Création d'un projet avec Apache Maven	200
1.3.2	Configuration Java	202
1.3.3	Création des objets du catalogue REST	207
1.3.4	Création des services REST	208
1.3.5	Génération du WADL	209
1.3.6	Utilisation des services	210
1.4	Play Framework	212
1.4.1	Console Play	213
1.4.2	Création d'une application	213
1.4.3	Démarrage du serveur	214
1.4.4	Développement sous Eclipse	215
1.4.5	Compilation	216
1.4.6	Débogage	217
1.4.7	Ajout d'un service REST	217
2.	Mise en cache	220
2.1	Apache	220
2.1.1	Activation et désactivation d'un module	220
2.1.2	Module expires	222
2.1.3	Modules de cache	223

2.2	Les en-têtes HTTP	225
2.2.1	Valeurs transmises par le serveur	225
2.2.2	Valeurs transmises par le client	226
2.2.3	public, private, cache-extension	226
2.2.4	max-age, no-cache, no-store, smax-age	227
2.2.5	max-stale, min-fresh	227
2.2.6	no-transform	227
2.2.7	only-if-cached	227
2.2.8	must-revalidate, proxy-revalidate	228
2.3	Java	228
2.3.1	Installation et lancement de REDIS	228
2.3.2	Utilisation de REDIS avec Play	229
3.	Exemples d'API REST	235
3.1	Facebook	235
3.2	Twitter	239
3.2.1	Création d'une application Twitter	241
3.2.2	Test des requêtes présentées dans la documentation	245
4.	Restlet	248
4.1	Création d'un projet JEE	249
4.1.1	Modification du fichier pom.xml	249
4.1.2	Création du fichier web.xml	252
4.1.3	Création du fichier applicationContext.xml	253
4.1.4	Création de la classe ApplicationMusicale	253
4.2	Création d'un service	254
4.2.1	Référencement de l'URL d'appel	254
4.2.2	Création de la classe du service	254
4.2.3	Appel du service	255
4.3	Création d'un client	256
5.	HATEOAS	259

8 ————— Développer des services REST

en Java - Échanger des données au format JSON

Chapitre 6

Exceptions communes

1. Erreurs de bibliothèques et de dépendances	261
1.1 Erreurs sous Apache CXF	262
1.2 Erreurs de bibliothèques sous Jersey	276
1.3 Erreurs de bibliothèques sous JBoss RESTEasy	277
2. Erreurs de configuration et erreurs de code.	280
2.1 Erreurs sous Apache CXF	280
2.2 Erreurs sous Jersey	282
2.3 Erreurs sous JBoss RESTEasy	284
3. Bugs reconnus	289
3.1 Bugs rencontrés sous Jersey	289
3.2 Bugs rencontrés sous JBoss RESTEasy	290
4. Autres types d'erreurs	291
4.1 Erreurs de port	291

Webographie

1. Apache	293
2. Apache CXF	294
3. Apache HttpComponents	294
4. Apache Maven	294
5. Apache Tomcat	295
6. CSS	295
7. Cryptographie	295
8. Google Chrome	295
9. Eclipse	296
10. Facebook	296
11. HTTP	296

12. IANA	296
13. Internet Explorer	297
14. Java	297
15. JavaScript	298
16. Jersey	298
17. JBoss	299
18. JBoss RESTEasy	299
19. JSON	299
20. Maven	300
21. Mozilla	300
22. MySQL	301
23. OAuth	301
24. Opera	301
25. Play Framework	301
26. Redis	302
27. REST	302
28. Restlet	302
29. RFC	302
30. ROA	303
31. ROCA	303
32. Safari	303
33. Spring	303
34. Twitter	304
35. WebKit	304
Index	305



Chapitre 4

Production de JSON avec Java et REST

1. Du POJO au JSON grâce aux annotations JAX-RS

JAX-RS est l'acronyme pour *Java API for RESTful Services*.

Cette API (*Application Programming Interface*) met à disposition des annotations pour créer simplement des services REST.

1.1 Les bases

Une fois choisie et ajoutée dans votre projet, l'implémentation de JAX-RS, la première étape est de choisir la méthode HTTP qui appellera telle ou telle méthode de votre application.

1.1.1 Choix de la méthode HTTP

Bien que le protocole HTTP soit large, comme vu dans le chapitre Introduction à REST, les méthodes prises en charge dans JAX-RS se limitent aux méthodes OPTIONS, HEAD, GET, POST, PUT et DELETE. Vous ne pourrez donc pas utiliser PATCH, LINK, UNLINK, TRACE et CONNECT.

86 ————— Développer des services REST

en Java - Échanger des données au format JSON

Les annotations suivantes sont disponibles :

- `@OPTIONS` pour connaître les méthodes implémentées.
- `@HEAD` pour connaître les en-têtes d'une ressource.
- `@GET` pour récupérer une ressource.
- `@POST` pour créer une ressource.
- `@PUT` pour modifier une ressource.
- `@DELETE` pour supprimer une ressource.

Ces annotations sont à positionner dans la classe, non pas à la déclaration de la classe, mais à la déclaration de la méthode qui sera exécutée.

@HEAD

Il est à noter qu'il n'est pas obligatoire d'implémenter la méthode HEAD. Pour répondre à une requête HEAD envoyée au service REST, JAX-RS regarde s'il y a une méthode avec l'annotation `@HEAD`, et si la méthode n'existe pas, c'est la méthode avec l'annotation `@GET` qui est automatiquement exécutée, mais son corps ne sera pas renvoyé. Cependant, les performances peuvent être moindres dans ce cas.

@OPTIONS

Comme pour l'annotation `@HEAD`, il est possible mais non obligatoire d'implémenter manuellement la méthode OPTIONS à l'aide de l'annotation `@OPTIONS`. Si la méthode n'existe pas, le résultat sera généré automatiquement à l'aide des méthodes définies dans la classe appelée.

1.1.2 Choix du chemin d'appel

Une fois votre méthode choisie, il reste à déterminer l'URL qui sera appelée. Elle sera relative à votre application, il ne faut donc pas remettre ni le protocole, ni le nom de domaine, ni le chemin de contexte de l'application. Cela passe par l'utilisation de l'annotation `@Path` qui prend en paramètre le chemin d'appel.

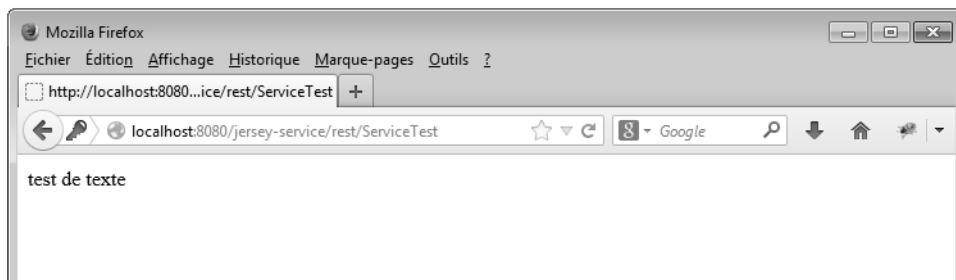
Cette annotation se positionne en deux endroits : obligatoirement à la déclaration de la classe, et de manière facultative à la déclaration de la méthode qui sera exécutée.

```
@Path("/ServiceTest")
public class ServiceTest {

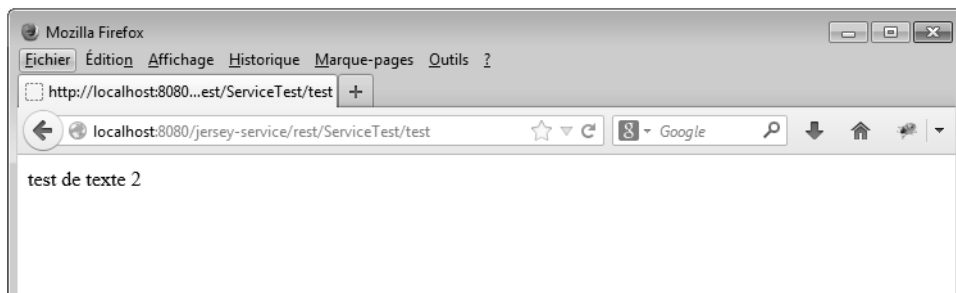
    @GET
    public String test() {
        return "test de texte";
    }

    @GET
    @Path("/test")
    public String test2() {
        return "test de texte 2";
    }
}
```

L'appel de /ServiceTest provoquera la sortie textuelle suivante.



L'appel de /ServiceTest/test provoquera la sortie textuelle suivante.



88 ————— Développer des services REST

en Java - Échanger des données au format JSON

1.1.3 Choix du type de retour

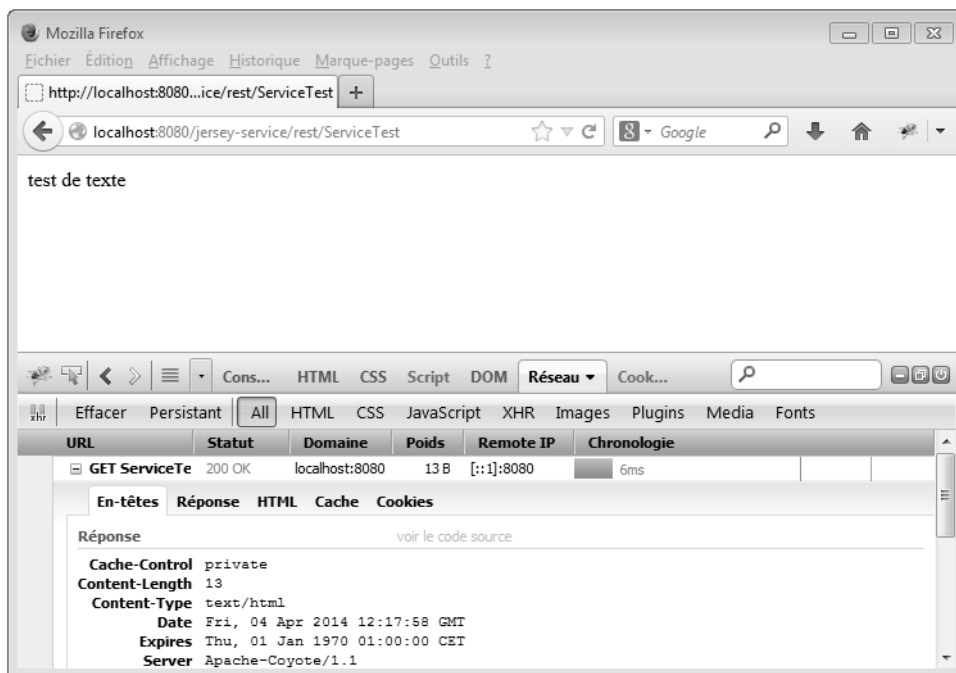
Après avoir choisi la méthode HTTP et l'URL relative d'appel, il faut choisir le type MIME de sortie.

Pour ce faire, il faut utiliser l'annotation `@Produces`, avec un attribut de la classe `MediaType` en paramètre.

Par défaut, si l'annotation n'est pas placée, du **text/html** sera généré.

```
@Path("/ServiceTest")
public class ServiceTest {

    @GET
    public String test() {
        return "test de texte";
    }
}
```



Dans l'exemple ci-après, le retour sera catégorisé JSON, grâce à la présence d'un argument, qui peut être une chaîne de caractères manuelle ou l'un des types prédéfinis dans la classe `MediaType`.

```
@Path("/ServiceTest")
public class ServiceTest {

    [...]

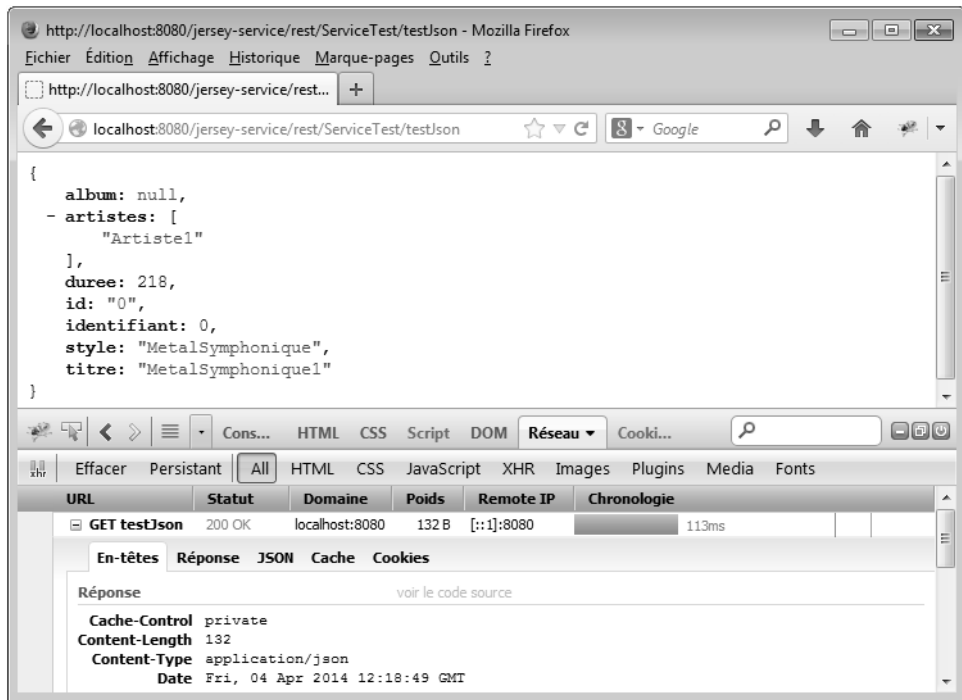
    @GET
    @Path("/testJson")
    @Produces(MediaType.APPLICATION_JSON)
    public String jsonTest() {
        final TitreMusique monTitreMusique
            = new TitreMusique("MetalSymphonique1", 218F);

        monTitreMusique.setStyle("MetalSymphonique");
        monTitreMusique.ajouteArtiste("Artiste1");

        final JSON json = new JSON(2);
        return json.format(monTitreMusique);
    }
}
```

90 ————— Développer des services REST

en Java - Échanger des données au format JSON



Il est possible de mettre plusieurs arguments, en les listant entre accolades. Le retour dépendra alors du type demandé par l'application appelante, grâce à l'en-tête HTTP Accept.

```
@Path("/ServiceTest")
public class ServiceTest {

    [...]

    @GET
    @Path("/testMultiple")
    @Produces({
        MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML
    })
    public String testMultiple() {
        final TitreMusique monTitreMusique
            = new TitreMusique("MetalSymphonique1", 218F);

        monTitreMusique.setStyle("MetalSymphonique");
    }
}
```